Otimização de Processos (COQ897)

Prof. Argimiro R. Secchi

Primeira Lista de Exercícios - 2020

José Rodrigues Torraca Neto

1) A engenheira Diana Prince (!), responsável por um determinado processo químico, notou, ainda na fase de projeto da planta, a ocorrência da reação de hidrogenação do eteno:

$$\begin{matrix} C_2H_4 & + & H_2 & \rightleftarrows & C_2H_6 \\ n_2 & & n_3 & & n_1 \end{matrix}$$

que para fins do processo em questão é indesejada. Querendo saber a quantidade de eteno que seria perdida no processo, Diana decidiu calcular o número de mols n_1 , n_2 e n_3 das espécies em equilíbrio, lembrando que, no equilíbrio, a energia de Gibbs total do sistema, $G_t($ n_1, n_2, n_3) é mínima. Sabendo que as espécies atômicas se conservam, qual foi o problema de otimização formulado pela Eng. Diana?

Solução:

O problema de otimização em questão é minimizar a função objetivo dada pela energia livre de Gibbs (G).

Para qualquer processo infinitesimal em que a quantidade de espécies presente pode ser alterada pela transferência de espécies de/para uma fase ou por reação química, o diferencial da energia livre de Gibbs é dado por:

$$dG = SdT + VdP + \sum_{i=1}^{n} \mu_i dn_i \qquad (1)$$

Onde G, S, T e P são: a energia livre de Gibbs, a entropia, a temperatura e a pressão (total), respectivamente. A energia livre molal parcial da espécie i é μ_i (potencial químico), e n_i é o número de moles da espécie i no sistema.

Se for assumido que a temperatura e a pressão são mantidas constantes durante o processo, dT e dP desaparecem. Se agora fizermos alterações em n_i de modo que $dn_i = dkn_i$, com as variações em n_i na mesma proporção k; então, uma vez que G é uma quantidade extensiva, devemos ter $dG = dkG$. Isso implica que:

$$G = \sum_{i=1}^{n} \mu_i n_i \qquad (2)$$

A comparação das Equações (1) e (2) mostra que os potenciais químicos são quantidades intensivas, ou seja, não dependem da quantidade de cada espécie, pois se todos os n_i são aumentados na mesma proporção com T e P constantes, μ_i deve permanecer inalterado para G aumentar na mesma taxa que n_i. Esta propriedade de invariância do μ_i é de extrema importância para restringir as formas possíveis que o μ_i pode assumir.

A equação (2) expressa a energia livre de Gibbs em termos dos números molares n_i, que aparecem explícita e implicitamente (no μ_i) no lado direito. A energia livre de Gibbs é mínima quando o sistema está em equilíbrio. O problema básico, então, torna-se o de encontrar aquele conjunto de n_i que torna G um mínimo.

Sendo n_i^* o número de moles dos compostos em equilíbrio e M (3) o número de elementos presentes no sistema, e presumindo que o número inicial de moles de cada composto é conhecido:

O problema consiste em (com T e P ctes):

$$Minimizar \quad G = \sum_{i=1}^{M=3} (\mu_i^o + RTlnP + RTlnx_i)n_i$$

$$G = RTlnP + [\sum_i \mu_i^o + RT \sum_i lnx_i](n_i)$$

$$, \; com \quad RTlnP = cte, \quad \mu_i^o = \sum_i RTlnK_x, \quad x_i = \frac{n_i}{n} = \frac{n_i}{\sum n_i}$$

$$, \; sujeito \; ao \; balanço \; estequiométrico :$$

$$\sum_i a_{ik} n_i = b_k, \quad para \; cada \; um \; dos \; elementos \; k = 1 \ldots M(=3)$$

$$e \; restrições \; de \; desigualdade :$$

$$n_i \geq 0$$

$$, \; com \quad n_i = x_i n.$$

$$Para \; (2) + (3) \rightleftarrows (1), \quad K_x = (\frac{n_1}{n_T})/(\frac{n_2}{n_T})(\frac{n_3}{n_T})$$

2) Dada a função objetivo $S(x_1, x_2) = 7,5x_1^2 + 12x_2^2 - 3x_1^2x_2^2 + 18x_1 + 11$, determine a localização e a natureza (mínimo, máximo ou sela) dos seus pontos estacionários. Esboce o gráfico da superfície da função objetivo em função de x_1 e x_2 e outro gráfico com 50 curvas de níveis, ambos contendo todos os pontos estacionários encontrados. Indique no segundo gráfico a localização dos pontos estacionários.

Solução:

$$S(x_1, x_2) = 7,5x_1^2 + 12x_2^2 - 3x_1^2x_2^2 + 18x_1 + 11$$

$$\nabla S(x_1, x_2) = \begin{pmatrix} 15x_1 - 6x_1x_2^2 + 18 \\ 24x_2 - 6x_1^2x_2 \end{pmatrix}$$

$$Então, para \; encontrar \; o \; ponto \; ótimo \; x^*(x_1, x_2) \; em \; que \; \nabla S(x_1, x_2) = 0 :$$

```
import numpy as np
import scipy.integrate
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.optimize

# definindo o sistema de equações como uma função do Python

def func (x):
    return [15.0*x[0] - (6.0*x[0])*(x[1])**2. + 18.0,
            24.0*x[1] - (6.0*x[1])*(x[0])**2.]

# estimativa inicial

x0 = [0, 0]

# resolvendo

result = scipy.optimize.root(func, x0)

# imprimindo resultado

print(result)

print(result.x)
```

```
         fjac: array([[-1.,  0.],
            [ 0., -1.]])
          fun: array([0., 0.])
      message: 'The solution converged.'
         nfev: 5
          qtf: array([3.92610389e-11, 0.00000000e+00])
            r: array([-15.,   0., -24.])
       status: 1
      success: True
            x: array([-1.2,  0. ])
     [-1.2  0. ]
```

```
def func (x):
    return [15.0*x[0] - (6.0*x[0])*(x[1])**2. + 18.0,
            24.0*x[1] - (6.0*x[1])*(x[0])**2.]

# mudando a estimativa inicial

x0 = [-5, -5]

# resolvendo

result = scipy.optimize.root(func, x0)

# imprimindo resultado

print(result)

print(result.x)
```

```
         fjac: array([[-0.35114934,  0.93631946],
                [-0.93631946, -0.35114934]])
          fun: array([ 2.98427949e-13, -7.46069873e-13])
      message: 'The solution converged.'
         nfev: 18
          qtf: array([-2.74861469e-09, -5.09792308e-11])
            r: array([-25.6602389 ,   7.4602378 ,  22.44745608])
       status: 1
      success: True
            x: array([-2., -1.])
     [-2. -1.]
```

```
def func (x):
    return [15.0*x[0] - (6.0*x[0])*(x[1])**2. + 18.0,
            24.0*x[1] - (6.0*x[1])*(x[0])**2.]

# mudando a estimativa inicial

x0 = [4, 1]
```

```
x0 = [4, 1]

# resolvendo

result = scipy.optimize.root(func, x0)

# imprimindo resultado

print(result)

print(result.x)
```

```
         fjac: array([[ 0.11023129,  0.99390596],
            [-0.99390596,  0.11023129]])
          fun: array([ 3.55271368e-14, -2.13162821e-14])
      message: 'The solution converged.'
         nfev: 14
          qtf: array([7.30358317e-10, 3.00525848e-09])
            r: array([-49.58732703,  -5.59353329,  46.46266912])
       status: 1
      success: True
            x: array([2., 2.])
     [2. 2.]
```

```
def func (x):
    return [15.0*x[0] - (6.0*x[0])*(x[1])**2. + 18.0,
            24.0*x[1] - (6.0*x[1])*(x[0])**2.]

# mudando a estimativa inicial

x0 = [3, -3]

# resolvendo

result = scipy.optimize.root(func, x0)

# imprimindo resultado

print(result)
```

```
print(result.x)

         fjac: array([[-0.426927  ,  0.90428609],
            [-0.90428609, -0.426927  ]])
          fun: array([-2.94377855e-11, -2.13731255e-11])
      message: 'The solution converged.'
         nfev: 13
          qtf: array([ 1.06789821e-09, -5.66347549e-09])
            r: array([ 44.91346119, -19.06281987, -50.97105675])
       status: 1
      success: True
            x: array([ 2., -2.])
     [ 2. -2.]


def func (x):
    return [15.0*x[0] - (6.0*x[0])*(x[1])**2. + 18.0,
            24.0*x[1] - (6.0*x[1])*(x[0])**2.]

# mudando a estimativa inicial

x0 = [-3, 3]

# resolvendo

result = scipy.optimize.root(func, x0)

# imprimindo resultado

print(result)

print(result.x)

         fjac: array([[ 0.16400028,  0.98646029],
            [-0.98646029,  0.16400028]])
          fun: array([ 1.70530257e-13, -2.69650968e-12])
      message: 'The solution converged.'
         nfev: 16
```

```
     qtf: array([-1.08506504e-08, -2.51441215e-09])
       r: array([ 46.47932236,  52.61229264, -12.39589512])
  status: 1
 success: True
       x: array([-2.,  1.])
[-2.  1.]
```

$x^* = \begin{pmatrix} -1,2 \\ 0,0 \end{pmatrix}, \begin{pmatrix} -2,0 \\ -1,0 \end{pmatrix}, \begin{pmatrix} 2,0 \\ 2,0 \end{pmatrix} \begin{pmatrix} 2,0 \\ -2,0 \end{pmatrix} \begin{pmatrix} -2,0 \\ 1,0 \end{pmatrix}$

$Calculando\ a\ matriz\ Hessiana:$

$H(x) = \begin{pmatrix} 15 - 6x_2^2 & -12x_1x_2 \\ -12x_1x_2 & 24 - 6x_1^2 \end{pmatrix}$

$No\ ponto\ \text{ótimo}\ x^* = \begin{pmatrix} -1,2 \\ 0,0 \end{pmatrix}:$

$H(x^*) = \begin{pmatrix} 15 & 0 \\ 0 & 15,36 \end{pmatrix}$

$Logo,\ a\ matriz\ H(x^*)\ \text{é}\ positiva\ definida\ neste\ ponto,\ o\ que\ implica\ em\ x^* = \begin{pmatrix} -1,2 \\ 0,0 \end{pmatrix}\ ser\ um\ ponto\ de\ mínimo\ local.$

$No\ ponto\ \text{ótimo}\ x^* = \begin{pmatrix} -2,0 \\ -1,0 \end{pmatrix}:$

$H(x^*) = \begin{pmatrix} 9 & -24 \\ -24 & 0 \end{pmatrix}$

$Como\ a\ matriz\ H(x^*)\ \text{não é}\ diagonal\ neste\ ponto,\ temos\ que\ calcular\ seus\ autovalores\ (\lambda):$

```
B = np.array([[9, -24],
              [-24, 0]])

sigma = np.linalg.eigvals(B)
sigma
```

```
array([ 28.91823089, -19.91823089])
```

> $\lambda = \begin{pmatrix} 28,9 \\ -19,9 \end{pmatrix}$
>
> $Logo,\ a\ matriz\ H(x^*)$ não é $definida\ neste\ ponto,\ o\ que\ implica\ em\ x^* = \begin{pmatrix} -2,0 \\ -1,0 \end{pmatrix}\ ser\ um\ ponto\ de$
>
> $sela.$

> $No\ ponto$ ótimo $x^* = \begin{pmatrix} 2,0 \\ 2,0 \end{pmatrix}:$
>
> $H(x^*) = \begin{pmatrix} -9 & -48 \\ -48 & 0 \end{pmatrix}$
>
> $Como\ a\ matriz\ H(x^*)$ não é $diagonal\ neste\ ponto,\ temos\ que\ calcular\ seus\ autovalores\ (\lambda):$

```
C = np.array([[-9, -48],
              [-48, 0]])

sigma = np.linalg.eigvals(C)
sigma
```

```
array([-52.71047604,  43.71047604])
```

> $\lambda = \begin{pmatrix} -52,7 \\ 43,7 \end{pmatrix}$
>
> *Logo, a matriz* $H(x^*)$ *não é definida neste ponto, o que implica em* $x^* = \begin{pmatrix} 2,0 \\ 2,0 \end{pmatrix}$ *ser um ponto de sela.*

> *No ponto ótimo* $x^* = \begin{pmatrix} 2,0 \\ -2,0 \end{pmatrix}$ *:*
>
> $H(x^*) = \begin{pmatrix} -9 & 48 \\ 48 & 0 \end{pmatrix}$
>
> *Como a matriz* $H(x^*)$ *não é diagonal neste ponto, temos que calcular seus autovalores* (λ) *:*

```
C = np.array([[-9, 48],
              [48, 0]])

sigma = np.linalg.eigvals(C)
sigma
```

```
array([-52.71047604,  43.71047604])
```

> $\lambda = \begin{pmatrix} -52,7 \\ 43,7 \end{pmatrix}$
>
> $Logo,\ a\ matriz\ H(x^*)$ não é $definida\ neste\ ponto,\ o\ que\ implica\ em\ x^* = \begin{pmatrix} 2,0 \\ -2,0 \end{pmatrix}\ ser\ um\ ponto\ de\ sela.$

> $No\ ponto$ ótimo $x^* = \begin{pmatrix} -2,0 \\ 1,0 \end{pmatrix}:$
>
> $H(x^*) = \begin{pmatrix} 9 & 24 \\ 24 & 0 \end{pmatrix}$
>
> $Como\ a\ matriz\ H(x^*)$ não é $diagonal\ neste\ ponto,\ temos\ que\ calcular\ seus\ autovalores\ (\lambda):$

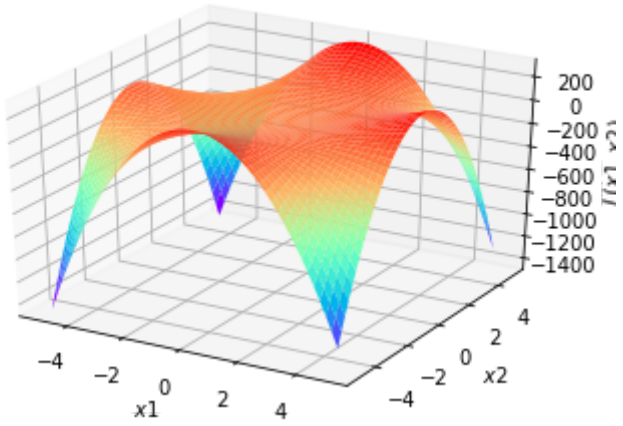
```
C = np.array([[9, 24],
              [24, 0]])

sigma = np.linalg.eigvals(C)
sigma
```

```
array([ 28.91823089, -19.91823089])
```

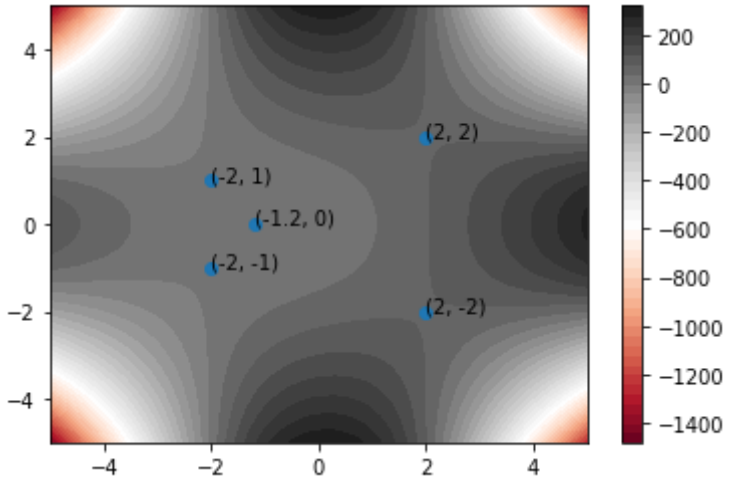
$$\lambda = \begin{pmatrix} 28,9 \end{pmatrix}$$

Plotando a superfície e as curvas de níveis:

```
#Plot surface 3d:

from matplotlib import cm

x1 = np.linspace(-5., 5., 50)
x2 = np.linspace(-5., 5., 50)
X, Y = np.meshgrid(x1, x2)

Z = 7.5*X**2 + 12*Y**2 - 3*X**2*Y**2 + 18*X + 11

fig, ax = plt.subplots(subplot_kw={'projection': '3d'})
ax.plot_surface(X, Y, Z, cmap=cm.rainbow)

ax.set_xlabel('$x1$')
ax.set_ylabel('$x2$')
ax.set_zlabel('$L(x1,x2)$');
```



```
#Plot density - contour (with colorbar) - with stationary points:

plt.contourf(X, Y, Z, 50, cmap='RdGy')
plt.colorbar();
plt.scatter([-1.2, -2., 2., 2., -2.], [0, -1., 2., -2, 1.])
plt.annotate("(-1.2, 0)", (-1.2, 0))
plt.annotate("(-2, -1)", (-2., -1.))
plt.annotate("(2, 2)", (2., 2.))
plt.annotate("(2, -2)", (2., -2.))
plt.annotate("(-2, 1)", (-2., 1.))
plt.show()
```




```
#Calculando o valor de S(x1,x2) nos pontos estacionários:

Z = 7.5*X**2 + 12*Y**2 - 3*X**2*Y**2 + 18*X + 11

def f(x11, x22):
    return 7.5*x11**2 + 12*x22**2 - 3*x11**2*x22**2 + 18*x11 + 11

result1 = f(-1.2, 0)
result2 = f(-2., -1.)
result3 = f(2., 2.)
```

```
print(result1, result2, result3)
```

```
0.20000000000000107 5.0 77.0
```

```
#Plot density - contour (with labels):

Z = 7.5*X**2 + 12*Y**2 - 3*X**2*Y**2 + 18*X + 11

fig, ax = plt.subplots()
CS = ax.contour(X, Y, Z, [0.2,5.0,77.0], cmap='jet')
ax.clabel(CS, inline=1, fontsize=10)
ax.set_title('Countour with labels')
ax.set_xlabel('$x1$')
ax.set_ylabel('$x2$')
```

```
Text(0, 0.5, '$x2$')
```